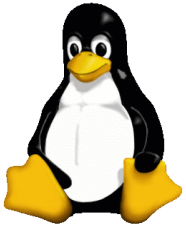


Diskless Carrier Grade Linux with Debian HPTE 2.0

Version 1.0

Chad N. Tindel <chad.tindel@hp.com>

Jan Ariyasu <jan.ariyasu@hp.com>



Open Source and Linux Organization

Table of Contents

1. Overview.....	3
2. How Network Booting Works.....	3
2.1 RAM Disk Architectures.....	4
2.2 NFS Root Architectures.....	5
3. Set Up The TFTP Server.....	6
3.1.1 TFTP Server Configuration for RAM disk Architectures on x86/x86_64 platforms.....	6
3.1.2 TFTP Server Configuration for NFS Root Architectures on x86/x86_64 platforms.....	6
3.2.1 TFTP Server Configuration for RAM disk Architectures on ia64 platforms.....	7
3.2.2 TFTP Server Configuration for NFS Root Architectures on ia64 platforms.....	7
4. Set Up The DHCP Server.....	8
5. Set Up The NFS Server.....	9
6. Creating the Network Boot Image.....	9
6.1 debian_netboot_create.sh usage.....	9
6.1.1 --debian-mirror (Required).....	10
6.1.2 --netboot-type (Required).....	10
6.1.3 --operation (Required).....	10
6.1.4 --tftp-location (Optional).....	10
6.1.5 --nfs-location (Optional).....	10
6.1.6 --ethernet-driver (Required only if --netboot-type is nfsroot).....	10
6.1.7 --debian-release (Optional).....	11
6.1.8 --apt-sources (Optional).....	11
6.1.9 --serial-console-device (Optional).....	11
6.1.10 --serial-console-speed (Optional).....	11
6.1.11 --serial-console-term-type (Optional).....	11
6.1.12 --initrd-tmp-size (Optional).....	11
6.1.13 --initrd-final-size (Optional).....	11
6.1.14 --post-bootstrap-hook (Optional).....	11
6.1.15 --post-image-creation-hook (Optional).....	12
6.1.16 --verbose.....	12
6.1.17 --help.....	12
6.2 Common variables and utilities.....	12
6.3 Creating the initrd Image.....	13
6.4 Disabling RC Scripts.....	14
6.5 Installing Base Applications.....	14
6.6 Set Up Security and Login Information.....	15

6.7 Set Up Local Hardware Discovery and Networking.....	15
6.8 Set Up Kernel Image and mkinitrd Settings.....	16
6.9 Install the HP Telco Kernel and Applications.....	17
6.10 Create Final Network Boot Image.....	17
6.11 Operation Handler Functions.....	18
6.11 Command Line Parsing and Operation Invocation.....	19
7. Conclusion.....	21

1. Overview

Of all the hardware components in a Carrier-Grade Telecom installation, hard disk drives (HDDs) are the most problematic and prone to failure. They are incredibly complicated mechanical systems in their own right, with platters that spin at thousands of revolutions per minute and platter reading mechanisms that swing back and forth across the platters. Some customers use diskless configurations to increase simplicity and reliability.

Eliminating HDDs also results in faster system boot times. For example, HP has seen a reduction in boot times by up to 25 seconds upon removal of the SCSI subsystem from the Linux kernel. In conjunction with the KEXEC fast-reboot package inside of Debian HPTE, this can result in reboot times of less than 5 seconds¹.

Another advantage of diskless booting is the simplified process for upgrading the system and installing new packages. Rolling out new images only involves upgrading images on the boot server; so depending on how you choose to architect your boot process, upgrading the boot image for thousands of servers can be boiled down to a single step.

This document discusses various methods of building network boot images. It assumes that you already have Debian with HP Telco Extensions (HPTE) version 2.0 installed onto a system that does have a local HDD. This system will be used for building the network boot images, and will be referred to as the "build server". In addition, this document gives examples of how to configure the DHCP and TFTP servers. When you are done reading this document you should have all the information you need to set up all elements of a network boot environment.

A script (`debian_netboot_create.sh`) is presented which can be used to build several different types of network boot images. This script may do everything you need, but more likely it will be used as a starting point for building your custom images. Feel free to modify it to suit your purposes. The script has been tested on the HP ATCA bc2100 Blades as well as the HP cx2600 systems using Debian HPTE 2.0.11 and later.²

The script is released under the terms of the MIT license, so you can feel free to use, modify, and redistribute it.

For updates and future versions of this document, or to download the `debian_netboot_create.sh` script, please visit <http://opensource.hp.com>.

2. How Network Booting Works

The industry standard method for network booting is called the **Preboot eXecution Environment (PXE, pronounced "pixie")**. When a system PXE boots, it broadcasts a DHCP request; the DHCP response contains 3 key pieces of information:

1. An IP address for the system to use
2. The address of a TFTP server which will provide a boot image
3. The path to the correct boot image on the TFTP server

The boot image supplied in Step 3 is a program which is the starting point of the PXE boot process; it begins by reading a configuration file on the TFTP server, and then it downloads a Linux kernel and an *Initial RAM disk*. Information about how to setup the TFTP server and configuration files will be given in section 3.

The Initial RAM disk, or *initrd*, is a Linux filesystem containing the necessary pieces for actually booting the system. A RAM disk is basically a piece of system memory (RAM) that simulates the role of a disk drive. Network booting for Debian HPTE always involves the use of an *initrd*, regardless of whether you choose the RAM disk architecture or the NFS Root architecture.

There are two basic questions that must be considered when deciding how to architect the way your systems boot:

1. Will each system have its own boot image or will several systems share one boot image?
2. Will the root filesystem reside in a RAM disk or will it be accessed via a Network Filesystem?

Depending on how you answer these questions, you can end up with one of several different architectures.

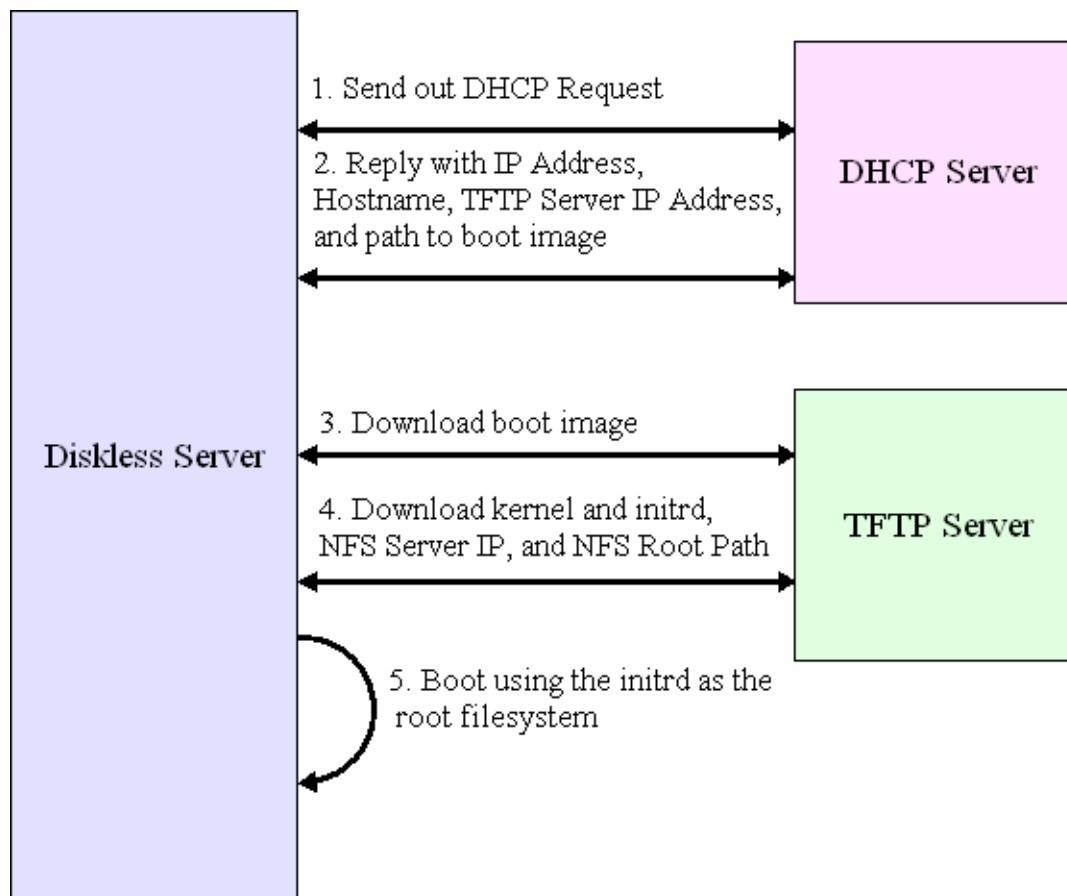
¹These two time measurements have been observed on IA-64 systems, and include the time from kernel load to the beginning of the *init* process

²Due to a bug in the x86_64 kernel of the 2.0.11 (2.6.10-telco-1.23-em64t-p4-smp), users wishing to use the x86_64 version of 2.0.11 must upgrade to update1 or later, for a kernel version of 2.6.10-telco-1.23.0.1-em64t-p4-smp or later.

2.1 RAM Disk Architectures

In a RAM disk architecture, the entire Debian installation is contained inside of the initrd. Once the initrd is loaded, the kernel will start the `/sbin/init` process and the system will boot normally. The script supplied with this document can be used to create an initrd image that can be used by multiple systems simultaneously, because all IP address and hostname configuration is done via DHCP.

The boot process for a RAM Disk Architecture is shown in the image below:



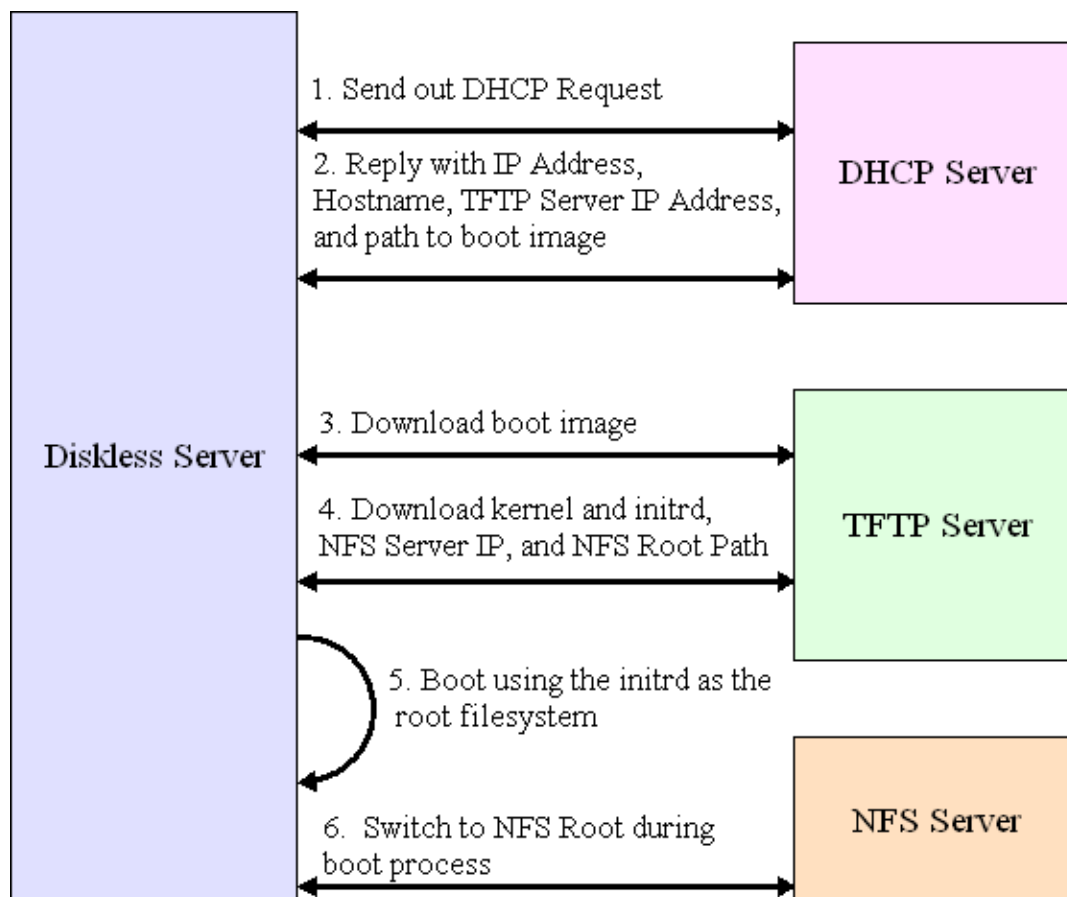
2.2 NFS Root Architectures

Two methods exist for doing an NFS Root architecture in Linux:

1. Build all the necessary components (Ethernet drivers, NFS drivers, NFS Root support, etc) into the kernel without using kernel modules.
2. Create an initrd that contains the necessary components in kernel modules form, and then mount the root filesystem via NFS from within the system startup sequence.

For Debian HPTE we have to use the 2nd method because Debian HPTE follows the Debian policy to use loadable modules whenever possible. We will make use of a Debian package called *lessdisks* which exists to enable this method of NFS Root.

The boot process for a NFS Root Architecture is shown in the image below:



3. Set Up The TFTP Server

This document assumes that your TFTP server is running Debian. Of course, this is not a hard requirement and you are free to use any system you wish.

Since the x86/x86_64 platforms use pxelinux as a netboot loader and the ia64 platform uses elilo.efi as a netboot loader, the instructions in this chapter are divided into different sections to cover each platform separately.

In addition, each platform has two possible architectures, a RAM disk architecture, or an NFS Root architecture. The instructions in each section are further divided to cover each architecture separately. You need only perform the instructions from the section that applies to your server's particular platform/architecture combination.

3.1 x86/x86_64 TFTP Server Configuration

The x86/x86_64 platforms use pxelinux as a netboot loader. You can download this program from H. Peter Anvin's website at <http://syslinux.zytor.com/pxe.php> or you can copy the pxelinux.0 binary out of one of the official Debian sarge netboot images. One thing to note is that the Linux PXE boot client (pxelinux.0) uses the TSIZE option, so your TFTP daemon must support the TSIZE option.

The standard Debian TFTP daemon (from the *tftp* package) does not support the TSIZE option; you should instead use the *tftpd-hpa* package. Our examples will assume that the TFTP root directory is */tftpboot*, and that all of the Debian network boot images will exist in */tftpboot/debian_netboot*.

The example configuration files assume that your TFTP Server and (assuming you are using the NFS Root method) your NFS Server have an IP address of 10.15.10.30.

To configure your TFTP server, perform the instructions in either section 3.1.1 or 3.1.2, depending upon whether your server has a RAM disk architecture, or an NFS root architecture.

3.1.1 TFTP Server Configuration for RAM disk Architectures on x86/x86_64 platforms

If you are using the RAM disk Architecture outlined in 2.1, then you will need the following files on your TFTP server:

- ***/tftpboot/debian_netboot/initrd/x86/pxelinux.0*** – The pxelinux.0 boot loader
- ***/tftpboot/debian_netboot/initrd/x86/vmlinuz*** – The Debian HPTE Linux Kernel
- ***/tftpboot/debian_netboot/initrd/x86/debian_diskless_initrd.gz*** – The Debian HPTE image that will be used as the initrd.
- ***/tftpboot/debian_netboot/initrd/x86/pxelinux.cfg/default*** – The pxelinux.0 configuration file

The contents of the *pxelinux.cfg/default* file need to be something like this:

```
default linux
label linux
kernel vmlinuz
append vga=normal ramdisk_size=524288 console=ttyS0,115200n8 ip=dhcp initrd=debian_diskless_initrd.gz
root=/dev/ram0 rw
```

It is important that everything on that last line is actually contained on one line in the file. You might be varying the *ramdisk_size=524288* option or *console=ttyS0,115200n8* option depending on your system configuration.

3.1.2 TFTP Server Configuration for NFS Root Architectures on x86/x86_64 platforms

If you are using the NFS Root Architecture outlined in 2.2, then you will need the following files on your TFTP server:

- ***/tftpboot/debian_netboot/nfs/x86/pxelinux.0*** – The pxelinux.0 boot loader
- ***/tftpboot/debian_netboot/nfs/x86/vmlinuz*** – The Debian HPTE Linux Kernel
- ***/tftpboot/debian_netboot/nfs/x86/initrd-img.nfsroot*** – The initrd image that will be used to mount the actual NFS Root filesystem.
- ***/tftpboot/debian_netboot/nfs/x86/pxelinux.cfg/default*** – The pxelinux.0 configuration file

The contents of the pxelinux.cfg/default file need to be something like this:

```
default linux
label linux
kernel vmlinuz
append vga=normal console=tty0 ip=dhcp initrd=initrd.img-nfsroot nfsroot=10.15.10.30:/nfsroot/debian_netboot
root=/dev/nfs rw
```

Again, it is important that everything on the last line is actually contained on one line in the file.

Note the parameter **nfsroot=10.15.10.30:/nfsroot/debian_netboot**. You will need to change this to match your actual NFS Server configuration.

3.2 IA-64 TFTP Server Configuration

The ia64 platforms use elilo.efi as a netboot loader. This program is available at /usr/lib/elilo/elilo.efi on any debian system with the *elilo* package installed.

Our examples will assume that the TFTP root directory is /tftpboot, and that all of the Debian network boot images will exist in /tftpboot/debian_netboot.

The example configuration files assume that your TFTP Server and (assuming you are using the NFS Root method) your NFS Server have an IP address of 10.15.10.30.

To configure your TFTP server, perform the instructions in either section 3.2.1 or 3.2.2, depending upon whether your server has a RAM disk architecture, or an NFS root architecture.

3.2.1 TFTP Server Configuration for RAM disk Architectures on ia64 platforms

If you are using the RAM disk Architecture outlined in 2.1, then you will need the following files on your TFTP server:

- **/tftpboot/debian_netboot/initrd/ia64/elilo.efi** – The elilo.efi boot loader
- **/tftpboot/debian_netboot/initrd/ia64/vmlinuz** – The Debian HPTE Linux Kernel
- **/tftpboot/debian_netboot/initrd/ia64/debian_diskless_initrd.gz** – The Debian HPTE image that will be used as the initrd.
- **/tftpboot/debian_netboot/initrd/ia64/elilo.conf** – The elilo configuration file

The contents of the elilo.conf file need to be something like this:

```
default=Linux
image=/debian_netboot/initrd/ia64/vmlinuz
label=Linux
initrd=/debian_netboot/initrd/ia64/debian_diskless_initrd.gz
root=/dev/ram
append="ramdisk_size=524288 console=ttyS0,115200n8"
read-only
```

You might be varying the ramdisk_size=524288 option or console-ttyS0,115200n8 option depending on your system configuration.

3.2.2 TFTP Server Configuration for NFS Root Architectures on ia64 platforms

If you are using the NFS Root Architecture outlined in 2.2, then you will need the following files on your TFTP server:

- **/tftpboot/debian_netboot/nfs/ia64/elilo.efi** – The elilo.efi boot loader
- **/tftpboot/debian_netboot/nfs/ia64/vmlinuz** – The Debian HPTE Linux Kernel
- **/tftpboot/debian_netboot/nfs/ia64/initrd-img.nfsroot** – The initrd image that will be used to mount the actual NFS Root filesystem.
- **/tftpboot/debian_netboot/nfs/ia64/elilo.conf** – The elilo configuration file

The contents of the elilo.conf file need to be something like this:

```
default=Linux
image=/debian_netboot/nfs/ia64/vmlinuz
label=Linux
initrd=/debian_netboot/nfs/ia64/initrd.img-nfsroot
root=/dev/ram
append="nfsroot=10.15.10.30:/nfsroot/debian_netboot ramdisk_size=524288 console=ttyS0,115200n8"
read-only
```

Note the parameter **nfsroot=10.15.10.30:/nfsroot/debian_netboot**. You will need to change this to match your actual NFS Server configuration.

4. Set Up The DHCP Server

This document assumes that your DHCP server is running Debian. Just like the TFTP Server, this is not a hard requirement. As long as your DHCP server allows you to specify a TFTP server/boot image pair and the target system's hostname you should be fine.

Your `/etc/dhcpd.conf` configuration will need to have your site-specific network topology, so this document doesn't attempt to give a full `dhcpd` configuration example. However, you will need to have a section that looks like this:

```
group {
  allow booting;
  allow bootp;
  option domain-name "mydomain.com";
  option smtp-server 10.0.0.2;
  option ntp-servers 10.0.0.2;
  use-host-decl-names on;

  host x86_initrd {
    hardware ethernet 00:A0:A5:55:0B:40;
    fixed-address 10.15.10.61;
    next-server 10.15.10.30;
    filename "/debian_netboot/initrd/x86/pxelinux.0";
  }

  host x86_nfs {
    hardware ethernet 00:A0:A5:55:0B:44;
    fixed-address 10.15.10.61;
    next-server 10.15.10.30;
    filename "/debian_netboot/nfs/x86/pxelinux.0";
  }

  host ia64_initrd {
    hardware ethernet 00:A0:A5:55:0B:48;
    fixed-address 10.15.10.63;
    next-server 10.15.10.30;
    filename "/debian_netboot/initrd/ia64/elilo.efi";
  }

  host ia64_nfs {
    hardware ethernet 00:A0:A5:55:0B:50;
    fixed-address 10.15.10.64;
    next-server 10.15.10.30;
    filename "/debian_netboot/nfs/ia64/elilo.efi";
  }
}
```

These 4 example servers will be assigned hostnames of `x86_initrd.mydomain.com`, `x86_nfs.mydomain.com`, `ia64_initrd.mydomain.com`, and `ia64_nfs.mydomain.com`. Obviously you'll want to fill in the appropriate site-specific network information like domain-name, MAC addresses, IP addresses, etc.

5. Set Up The NFS Server

If the server you plan to use is running Debian, then setting it up is trivial. Just run `apt-get install nfs-user-server`, and it will be ready to run. Following the example given in section 3.1.2, the following line would need to be in your `/etc/exports` file:

```
/nfsroot/debian_netboot 10.0.0.0/255.0.0.0(rw,no_root_squash,no_all_squash)
```

Just change the IP Network/Netmask values to match your site-specific network information.

6. Creating the Network Boot Image

This section discusses the creation of Network Boot Images. We will be defining, one piece at a time, a Bourne Shell script (**`debian_netboot_create.sh`**) which can be used for creating these network boot images. This script will create a generic Debian HPTE installation with the customizations required for either the RAM disk or NFS Root architectures.

In order to run `debian_netboot_create.sh`, HPTE must be installed on a system that is identical to your target netboot system (same processor family, same version of HPTE). We will refer to the system where you are running `debian_netboot_create.sh` as the *build server*. In addition, it assumes that the build server has an `/etc/apt/sources.list` containing the proper entries for installing Debian HPTE. These might point to a Debian HPTE mirror or to a local CD-ROM drive.

Building the netboot image is a two-step process. In the first step, a temporary filesystem is created and mounted via a loopback device. Debian HPTE and a base set of applications are installed into this temporary space, and the `debian_netboot_create.sh` will execute the user's Post-Bootstrap script if one was specified. This is a convenient time for installing any site-specific applications, configuration files, or data. Once everything is ready, any temporary files are removed.

In the second step, another filesystem is created and mounted via the loopback device. All the files from the temporary filesystem are copied into this finalized filesystem, and the finalized filesystem is compressed. At this point, `debian_netboot_create.sh` will execute the user's Post-Image-Creation script if one was specified. This can be used for distributing the images to various TFTP and NFS servers if they happen to be different systems than the build server.

The reason 2-steps are used is that the temporary filesystem needs to be significantly bigger than the finalized image. This allows us to have a much bigger working space while the Debian HPTE installation is being configured.

The rest of Section 6 describes the usage and various pieces of the `debian_netboot_create.sh` script.

6.1 `debian_netboot_create.sh` usage

The first function we define for our script is the `usage()` function. A subsection is given to describe each command line option and the default value for that option, if one exists.

```
usage()
{
    message="$1"
    exit_code="$2"

    if [ -n "$message" ]; then
        echo "Usage Error: $message"
        echo ""
    fi
    echo "Usage: debian_netboot_create.sh "
    echo "    --debian-mirror <Debian Mirror>"
    echo "    --netboot-type <initrd | nfsroot> "
    echo "    --operation <create | cleanup>"
}
```

```

echo "    [--tftp-location <TFTP Location>] "
echo "    [--nfs-location <NFS Location>]"
echo "    [--ethernet-module <ethernet driver>]"
echo "    [--debian-release <Debian Release>] "
echo "    [--apt-sources <APT sources.list file>]"
echo "    [--serial-console-device <Serial Console Device>] "
echo "    [--serial-console-speed <Serial Console Speed>] "
echo "    [--serial-console-term-type <Serial Console Term Type>] "
echo "    [--initrd-tmp-size <Temporary Image Size>] "
echo "    [--initrd-final-size <Final Image Size>] "
echo "    [--post-bootstrap-hook <Post Bootstrap Hook Script>] "
echo "    [--post-image-creation-hook <Post Image Creation Hook Script>]"
echo "    [--verbose]"
echo "    [--help]"

```

```

exit $exit_code
}

```

6.1.1 --debian-mirror (Required)

The --debian-mirror option gives the location of a Debian mirror which can be used by the debootstrap command.

Example: --debian-mirror <http://ftp.debian.org/debian>

6.1.2 --netboot-type (Required)

The --netboot-type option tells the script what method of netbooting you will be using your image for.

Allowed values: "initrd", "nfsroot"

6.1.3 --operation (Required)

The --operation option tells the script whether you are creating a new image or cleaning up files from the last build operation.

Allowed values: "create", "cleanup"

6.1.4 --tftp-location (Optional)

The --tftp-location option can be used if your build server is also functioning as your TFTP server. If you use this option, `debian_netboot_create.sh` will automatically copy the Linux kernel and `initrd` to the specified location.

Example: --tftp-location `/tftpboot/debian_netboot`

6.1.5 --nfs-location (Optional)

The --nfs-location option can be used if your build server is also functioning as your TFTP server. If you use this option, `debian_netboot_create.sh` will automatically copy the Debian Installation that was created by the script to the specified location.

Example: --nfs-location `/nfsroot/debian_netboot`

6.1.6 --ethernet-driver (Required only if --netboot-type is nfsroot)

If your netboot type is NFS Root, then `debian_netboot_create.sh` needs to know what the target system's ethernet driver is so that it can install the appropriate kernel module into the `initrd`. If your system needs multiple ethernet drivers, you can specify this option multiple times

Example: --ethernet-driver `e1000` --ethernet-driver `tg3`

6.1.7 --debian-release (Optional)

The --debian-release option is really only here as a placeholder for future versions of the scripts. It allows you to define differing Debian base versions for your netboot image.

Default value: sarge

6.1.8 --apt-sources (Optional)

If you wish to use a different APT sources.list file for building your image than the one currently used by your build server, you can specify this option.

Default value: /etc/apt/sources.list

6.1.9 --serial-console-device (Optional)

The --serial-console-device option allows you to specify the device file that should be used for the serial console.

Default value: ttyS0

6.1.10 --serial-console-speed (Optional)

The --serial-console-speed option allows you to specify the port speed of your serial console device.

Default value: 115200

6.1.11 --serial-console-term-type (Optional)

The --serial-console-term-type option allows you to specify the terminal type of your serial console device.

Default value: vt100

6.1.12 --initrd-tmp-size (Optional)

The --initrd-tmp-size parameter allows you to specify the size in KB of the temporary image where Debian is installed. Because lots of files go into this image before the image is cleaned up and finalized, it needs to be considerably larger than the final image size. If the default value of 1GB is not big enough for you, this parameter allows you to make it larger.

Default value: 1048576

6.1.13 --initrd-final-size (Optional)

If your netboot type is the RAM disk (initrd) method, the --initrd-final-size option allows you to specify the final (uncompressed) size of your initrd in KB. This value needs to be a power of 2.

Default value: 524288

If you decide that your image needs to be smaller than the default, keep in mind that you may need to use a Post-Bootstrap Hook script to remove things from the temporary image.

6.1.14 --post-bootstrap-hook (Optional)

The --post-bootstrap-hook option provides a convenient callout mechanism for the user to customize the debian installation. It is run right after the base applications and kernel are installed, and while the temporary filesystem is still in use. What is provided to the command line option is the path to a script that will be called at the appropriate time. Supplied to the callout script will be the following command line parameter:

- **--root <Debian Root Directory>**. The root directory is full path to the beginning of the temporary filesystem where Debian HPTE is being installed and configured. The provided Post-Bootstrap script is not executed inside of a chroot environment, so it will need to do that itself if necessary.

Example: `--post-bootstrap-hook /usr/local/bin/debian_netboot_create_post_bootstrap.sh`

6.1.15 --post-image-creation-hook (Optional)

The `--post-image-creation-hook` option provides a convenient callout mechanism for the user to distribute the finalized Debian HPTE images once they have been created. It is run right after the finalized image has been compressed. What is provided to the command line option is the path to a script that will be called at the appropriate time. Supplied to the callout script will be the following command line parameter:

- **--image-file <EXT3 filesystem image file>**. The image file is the finalized, uncompressed Debian HPTE installation. It can be copied to the appropriate NFS or TFTP servers if they are different than the build server.
 - In the case of the RAM disk architecture, this is the actual initrd that will be booted and used as a RAM disk by the target system. You will need mount this image somewhere and copy the `/vmlinuz` file to the TFTP server so that `pxelinux.0` can find them at boot time.
 - In the case of the NFS Root architecture, this is the root filesystem that needs to be in place on the NFS server. You will need to mount this image somewhere and copy the `/vmlinuz` and `/initrd.img` files to the TFTP server so that `pxelinux.0` can find them at boot time.
- **--image-file-compressed <gzip compressed EXT3 filesystem image file>**. Simply a compressed version of the same image passed to the `--image-file` option of the callout script.

Example: `--post-image-creation-hook /usr/local/bin/debian_netboot_create_post_image_creation.sh`

6.1.16 --verbose

Directs `debian_netboot_create.sh` to print out verbose informational and error messages.

6.1.17 --help

Directs `debian_netboot_create.sh` to print out the script usage and help display.

6.2 Common variables and utilities

All of the examples in this document use the Debian utility “`debootstrap`” to create a base Debian image that is manageable via `dpkg` and `apt`. This image is essentially an `initrd` which is a minimal image that allows you to boot and create whatever environment you want. The build script uses the following global variables utility functions during the build process:

```
BUILD_ROOT=`pwd`
LOOP1=$BUILD_ROOT/mnt/loop1
LOOP2=$BUILD_ROOT/mnt/loop2
INITRD=$BUILD_ROOT/debian_diskless_initrd
INITRD_TMP=$INITRD.tmp
INITRD_NFSROOT="$INITRD"_nfsroot
INITRD_NFSROOT_GZ="$INITRD_NFSROOT".gz
INITRD_GZ=$INITRD.gz
INITRD_TMP_SIZE=1048576
INITRD_FINAL_SIZE=524288
DEBIAN_MIRROR=""
DEBIAN_RELEASE=sarge
APT_SOURCES_FILE=/etc/apt/sources.list
SERIAL_CONSOLE_DEVICE=ttyS0
SERIAL_CONSOLE_SPEED=115200
SERIAL_CONSOLE_TERM_TYPE=vt100
POST_BOOTSTRAP_HOOK=""
POST_IMAGE_CREATION_HOOK=""
OPERATION=""
```

```

NETBOOT_TYPE=""
declare -a ETHERNET_MODULES
TFTP_LOCATION=""
NFS_LOCATION=""
VERBOSE=0
ARCH="" # x86 or ia64
KERNEL_VERSION=""

# Run and Check Command
rcc()
{
    command=$1
    if [ "$VERBOSE" -eq "1" ]
    then
        echo "$command"
        eval $command
        exit_code=$?
    else
        output=`$command`
        exit_code=$?
    fi

    if [ "$exit_code" -ne "0" ]
    then
        echo "Command \"$command\" failed with exit_code $exit_code"
        exit $exit_code
        if [ "$VERBOSE" -eq "1" ]
        then
            echo "Command \"$command\" failed with output:"
            echo $output
        fi
    fi
}

umount_loop_filesystems()
{
    fuser -kuv $LOOP1 > /dev/null 2>&1
    fuser -kuv $LOOP2 > /dev/null 2>&1
    umount $LOOP1/proc > /dev/null 2>&1
    umount $LOOP2/proc > /dev/null 2>&1
    umount /dev/loop1 > /dev/null 2>&1
    losetup -d /dev/loop1 > /dev/null 2>&1
    umount /dev/loop2 > /dev/null 2>&1
    losetup -d /dev/loop2 > /dev/null 2>&1
}

```

The process of creating the initrd image is two stages. In the first stage we use a bigger than necessary filesystem (initrd_tmp_SIZE) because of debian packages and other temporary files that need to exist. All installation and set up is done in this stage. In the second stage we clean up the image and copy all of the files from the temporary image into the final image. The size of the final image is given by initrd_final_size and must be a power of 2 in order to network boot properly.

6.3 Creating the initrd Image

The first thing that needs to be done is to create a file on disk that contains a Linux filesystem. This document uses the EXT3 filesystem type because it is linked into the Debian HPTE kernel (i.e. it is not built as a kernel module). The loop device is used to expand this filesystem-in-a-file onto a local mount point.

```

create_initrd_tmp_image()
{

```

```

rcc "mkdir -p $LOOP1"
rcc "mkdir -p $LOOP2"
rcc "dd if=/dev/zero of=$INITRD_TMP bs=1k count=$INITRD_TMP_SIZE"
rcc "mkfs.ext3 -F -L ROOT -m0 $INITRD_TMP"
rcc "losetup /dev/loop1 $INITRD_TMP"
rcc "mount -t ext3 /dev/loop1 $LOOP1"
echo "Creating $DEBIAN_RELEASE Bootstrap using mirror $DEBIAN_MIRROR..."
rcc "debootstrap $DEBIAN_RELEASE $LOOP1 $DEBIAN_MIRROR"
}

```

6.4 Disabling RC Scripts

When we get to the point where we are installing Debian packages into the Debian image, we don't want that package's control scripts to execute the RC script(s) installed by the package. These scripts might do things that start daemons; we are only building the images, and there is no need to start the daemons on the build machine. For the duration of the script that takes care of installing applications and the kernel (including the external callout scripts), RC scripts will be disabled inside of the chroot environment.

```

disable_rc_scripts()
{
    rcc "chroot $LOOP1 mv -f /usr/sbin/invoke-rc.d /usr/sbin/invoke-rc.d.old"
    rcc "chroot $LOOP1 ln -s /bin/true /usr/sbin/invoke-rc.d"
}

```

```

enable_rc_scripts()
{
    rcc "chroot $LOOP1 mv -f /usr/sbin/invoke-rc.d.old /usr/sbin/invoke-rc.d"
}

```

6.5 Installing Base Applications

Once a base Debian image exists, we need to install a base set of applications that are necessary to make the system usable. Examples are things like a DHCP client, USB utilities, SSH, etc. Note that a `/proc` filesystem needs to be mounted under the loop mountpoint in order for certain package to install properly. The reason that we have to use "dhcp3-client" instead of "dhcp-client" is because the old DHCP client package doesn't support the feature wherein the hostname can be set via DHCP. This is a good section for you to install other applications that you want in your system.

When we run `apt-get` we set the `DEBIAN_FRONTEND=noninteractive` and we use the "-y" option. This will ensure that the installer will not prompt us with installation questions for each package; it will just use the default answer.

```

install_base_applications()
{
    rcc "mount -tproc none $LOOP1/proc"
    rcc "cp -f $APT_SOURCES_FILE $LOOP1/etc/apt"
    rcc "chroot $LOOP1 apt-get update"

    # Remove the old dhcp client
    rcc "chroot $LOOP1 env DEBIAN_FRONTEND=noninteractive \
        apt-get -y remove --purge dhcp-client"

    pkgs="acpid initrd-tools coreutils fileutils module-init-tools less ssh \
        discover1 hotplug dhcp3-client tftp-hpa telnetd udev"

    if [[ "$ARCH" == "x86" ]]
    then
        pkgs="$pkgs lilo"
    fi

    rcc "chroot $LOOP1 env DEBIAN_FRONTEND=noninteractive apt-get -y install \
        $pkgs"
}

```

```

# Install packages that are only needed for NFS-Root mode
if [ "$NETBOOT_TYPE" == "nfsroot" ]; then
    rcc "chroot $LOOP1 env DEBIAN_FRONTEND=noninteractive \
        apt-get -y install initrd-netboot-tools udhpc"
fi
}

```

6.6 Set Up Security and Login Information

This example sets the root password to "root" (without quotes). Obviously you will want to set a more appropriate root password for your installations.

```

setup_security()
{
    # Setup the root password. The string cB/LVkpvmhEzE is an encrypted
    # password for "root". So login name is "root" (w/o quotes), and
    # password is the same.
    rcc "cat $LOOP1/etc/passwd | sed -e 's/^root:./root:cB/LVkpvmhEzE:/' > \
        $LOOP1/etc/passwd.tmp"
    rcc "mv -f $LOOP1/etc/passwd.tmp $LOOP1/etc/passwd"

    # Setup /etc/inittab with a serial port login
    cat >> $LOOP1/etc/inittab << EOF
T3:2345:respawn:/sbin/getty -L \
$SERIAL_CONSOLE_DEVICE $SERIAL_CONSOLE_SPEED $SERIAL_CONSOLE_TERM_TYPE
EOF
}

```

6.7 Set Up Local Hardware Discovery and Networking

```

setup_hardware_discovery()
{
    # Setup /etc/discover.conf
    cat > $LOOP1/etc/discover.conf <<EOF
enable pci,usb,ide,scsi
enable pcmcia
enable isa
disable parallel,serial
boot all
skip i810-tco
EOF
}

setup_networking()
{
    # Setup hosts stuff
    echo "127.0.0.1 localhost.localdomain localhost" > $LOOP1/etc/hosts

    # Setup /etc/network/interfaces
    cat > $LOOP1/etc/network/interfaces <<EOF
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
EOF
}

```

6.8 Set Up Kernel Image and mkinitrd Settings

Several files need to be configured before the telco kernel image can be installed into the initrd image because the control scripts will run a mkinitrd and we don't want that to fail.

```
setup_mkinitrd_conf()
{
    modules_string=$1

    cat > $LOOP1/etc/mkinitrd/mkinitrd.conf <<EOF
MODULES=$modules_string
DELAY=0
ROOT=
UMASK=022
MKIMAGE='mkcramfs %s %s > /dev/null'
BUSYBOX=no
PKGSCRIPTS=yes
INITRD_LD_LIBRARY_PATH=$LD_LIBRARY_PATH
EOF
}

setup_kernel_settings()
{
    # Setup /etc/kernel-img.conf
    # We need silent_modules=yes so that we can install the telco kernel
    # even if we are running the same version on the build machine. Without
    # this option we would get a prompt, causing the script to hang.
    cat > $LOOP1/etc/kernel-img.conf <<EOF
do_symlinks = yes
relative_links = yes
do_bootloader = no
do_bootfloppy = no
do_initrd = yes
link_in_boot = no
silent_modules = yes
EOF

    KERNEL_VERSION=`chroot $LOOP1 apt-cache depends task-hp-telco | \
        grep "Depends: kernel-image" | awk '{print $2;}' | \
        chroot $LOOP1 xargs apt-cache depends | \
        grep "Depends: kernel-image" | awk '{print $2;}' | \
        sed -e 's/kernel-image-//`

    # Setup mkinitrd
    if [ "$NETBOOT_TYPE" == "nfsroot" ]; then
        setup_mkinitrd_conf "dep"

        # If the user is going to do an NFS Root filesystem, we need to setup
        # the lessdisk configuration with the correct kernel modules path and
        # ethernet device driver name

        rcc "cat $LOOP1/etc/lessdisks/mkinitrd/initrd-netboot.conf | \
            sed -e 's/^nic_modules=.*\/nic_modules=\\\"${ETHERNET_MODULES[@]}\\\"/' > \
            $LOOP1/etc/lessdisks/mkinitrd/initrd-netboot.conf.tmp"
        rcc "echo \"MODULEDIR=/lib/modules/$KERNEL_VERSION\" >> \
            $LOOP1/etc/lessdisks/mkinitrd/initrd-netboot.conf.tmp"
        rcc "mv -f $LOOP1/etc/lessdisks/mkinitrd/initrd-netboot.conf.tmp \
            $LOOP1/etc/lessdisks/mkinitrd/initrd-netboot.conf"
    else
        setup_mkinitrd_conf "most"
    fi
}
```

```
fi
}
```

6.9 Install the HP Telco Kernel and Applications

This section installs the **task-hp-telco** package, which is a Debian Meta Package representing the Telco Kernel and all of the specialized Telco Applications. After the kernel installed, we need to shut down the various evlog daemons that were started on our system after the task-hp-telco package was installed.

Also, this section marks the end of the Debian HPTE Bootstrapping process. At the end of this section, the user's Post-Bootstrap callout is executed if one was specified.

```
install_kernel_image()
{
    rcc "chroot $LOOP1 env DEBIAN_FRONTEND=noninteractive \
        apt-get -y install task-hp-telco"

    # Run the user's post bootstrap hook script if they have one
    if [ -n "$POST_BOOTSTRAP_HOOK" ]; then
        rcc "$POST_BOOTSTRAP_HOOK --root $LOOP1"
    fi

    # Some architectures don't create an initrd by default
    if [ ! -f $LOOP1/initrd.img ]; then
        rcc "chroot $LOOP1 mkinitrd -o /initrd.img $KERNEL_VERSION"
    fi
}
```

6.10 Create Final Network Boot Image

This section is responsible for creating all the finalized, compressed image files. It will also distribute them to the appropriate locations if the build server also happens to be the NFS and TFTP servers (this is specified via the --tftp-location and --nfs-location command line options to debian_netboot_create.sh).

Also, this section marks the end of the image creation process. At the end of this section, the user's Post-Image-Creation callout is executed if one was specified.

```
finalize_images()
{
    rcc "chroot $LOOP1 apt-get clean"
    rcc "rm -f $LOOP1/etc/hostname"
    rcc "umount $LOOP1/proc"

    if [ "$NETBOOT_TYPE" == "nfsroot" ]; then
        initrd_final_name=$INITRD_NFSROOT
        initrd_final_name_gz=$INITRD_NFSROOT_GZ
        if [ -n "$NFS_LOCATION" ]; then
            mkdir -p $NFS_LOCATION
            rcc "cp -a $LOOP1/* $NFS_LOCATION"
        fi

        if [ -n "$TFTP_LOCATION" ]; then
            rcc "cp -f $LOOP1/vmlinuz $TFTP_LOCATION/vmlinuz-nfsroot"
            rcc "cp -f $LOOP1/initrd.img $TFTP_LOCATION/initrd.img-nfsroot"
        fi
    else
        initrd_final_name=$INITRD
        initrd_final_name_gz=$INITRD_GZ
        # NETBOOT_TYPE == initrd
        if [ -n "$TFTP_LOCATION" ]; then
```

```

    rcc "cp -f $LOOP1/vmlinuz $TFTP_LOCATION/vmlinuz"
fi
fi

# Regardless of the netboot type, we are going to create an initrd for
# them.
rcc "dd if=/dev/zero of=$initrd_final_name bs=1k count=$INITRD_FINAL_SIZE"
rcc "mkfs.ext3 -F -L ROOT -m0 $initrd_final_name"
rcc "losetup /dev/loop2 $initrd_final_name"
rcc "mount -t ext3 /dev/loop2 $LOOP2"
rcc "cp -a $LOOP1/* $LOOP2"
umount_loop_filesystems
rcc "fsck.ext3 -f $initrd_final_name"
rcc "gzip -c $initrd_final_name > $initrd_final_name_gz"

# If the mode is initrd we can copy the final initrd to the TFTP_LOCATION
if [ "$NETBOOT_TYPE" == "initrd" ]; then
    if [ -n "$TFTP_LOCATION" ]; then
        rcc "cp -f $initrd_final_name_gz $TFTP_LOCATION"
    fi
fi

# Run the user's post image creation hook script if they have one
if [ -n "$POST_IMAGE_CREATION_HOOK" ]; then
    rcc "$POST_IMAGE_CREATION_HOOK \
        --image-file $initrd_final_name \
        --image-file-gzipped $initrd_final_name_gz"
fi
}

```

6.11 Operation Handler Functions

This section defines the starting points for the various operations that can be specified on the command line.

```

create_image()
{
    case $NETBOOT_TYPE in
        initrd )
            if [ -n "$TFTP_LOCATION" ]; then
                echo "System Image initrd will be copied to TFTP Location: $TFTP_LOCATION"
            fi
            ;;
        nfsroot )
            if [ -n "$NFS_LOCATION" ]; then
                echo "System Image will be copied to NFS Location: $NFS_LOCATION"
            fi
            if [ -n "$TFTP_LOCATION" ]; then
                echo "NFS-Root initrd will be copied to TFTP Location: $TFTP_LOCATION"
            fi
            fi
            ;;

        if [ -z "$ETHERNET_MODULES" ]; then
            usage "NFS-Root netboot method requires the --ethernet-module option"
        else
            echo "Ethernet Drivers: ${ETHERNET_MODULES[@]}"
        fi
        ;;
        *)
            usage "Unknown netboot type $1" 1
    esac

    if [ -n "$DEBIAN_MIRROR" ]; then

```

```

    echo "Debian Mirror: $DEBIAN_MIRROR"
else
    usage "You must specify the --debian-mirror option" 1
fi

ARCH=`uname -m`
if [[ "$ARCH" == "i386" || "$ARCH" == "i486" ||
"$ARCH" == "i586" || "$ARCH" == "i686" || "$ARCH" == "x86_64" ]]
then
    ARCH="x86"
elif [ "$ARCH" != "ia64" ]
then
    usage "This script supports only on x86, x86_64, or ia64" 1
fi

create_initrd_tmp_image

disable_rc_scripts

install_base_applications

setup_security

setup_hardware_discovery

setup_networking

setup_kernel_settings

install_kernel_image

enable_rc_scripts

finalize_images
}

cleanup()
{
    umount_loop_filesystems

    rm -rf $LOOP1
    rm -rf $LOOP2
    rm -f $INITRD
    rm -f $INITRD_NFSROOT
    rm -f $INITRD_TMP
    rm -f $INITRD_GZ
    rm -f $INITRD_NFSROOT_GZ
    if [ -n "$NFS_LOCATION" ]; then
        rm -rf $NFS_LOCATION
    fi
}

```

6.11 Command Line Parsing and Operation Invocation

At this point all the important logic in the script is complete. All that remains is to parse the command line options and call the appropriate operation handler.

```

while [ "$1" != "" ]; do
    case $1 in
        --debian-mirror )

```

```

    shift
    DEBIAN_MIRROR=$1
    ;;
--debian-release )
    shift
    DEBIAN_RELEASE=$1
    ;;
--apt-sources )
    shift
    APT_SOURCES_PATH=$1
    ;;
--operation )
    shift
    OPERATION=$1
    ;;
--netboot-type )
    shift
    NETBOOT_TYPE=$1
    ;;
--tftp-location )
    shift
    TFTP_LOCATION=$1
    ;;
--nfs-location )
    shift
    NFS_LOCATION=$1
    ;;
--ethernet-module )
    shift
    ETHERNET_MODULES=( "${ETHERNET_MODULES[@]}" "$1" )
    ;;
--initrd-tmp-size )
    shift
    INITRD_TMP_SIZE=$1
    ;;
--initrd-final-size )
    shift
    INITRD_FINAL_SIZE=$1
    ;;
--serial-console-device )
    shift
    SERIAL_CONSOLE_DEVICE=$1
    ;;
--serial-console-speed )
    shift
    SERIAL_CONSOLE_SPEED=$1
    ;;
--serial-console-term-type )
    shift
    SERIAL_CONSOLE_TERM_TYPE=$1
    ;;
--post-bootstrap-hook )
    shift
    POST_BOOTSTRAP_HOOK=$1
    ;;
--post-image-creation-hook )      shift
    POST_IMAGE_CREATION_HOOK=$1
    ;;
-h | --help )
    usage "" 0

```

```

;;
-v | --verbose )
  VERBOSE=1
;;
*) usage "Unknown parameter $1" 1
esac
shift
done

case $OPERATION in
  create )
    create_image
    ;;
  cleanup )
    cleanup
    ;;
  *)
    usage "Unknown operation $1" 1
esac

```

7. Conclusion

By this point you should understand the concepts and mechanics of network booting. The script presented in this paper will provide you with a starting point for building your customized network boot images. Feel free to change it, modify it, and distribute to make it work for you.

For updates and future versions of this document, or to download the `debian_netboot_create.sh` script, please visit <http://opensource.hp.com>.